An Ontology Design Pattern for IoT Device Tagging Systems

Victor Charpenay*†, Sebastian Käbisch*, Darko Anicic* and Harald Kosch†
*Siemens AG — Corporate Technology
Email: *firstname.lastname*@siemens.com

†Universität Passau — Fakultät für Informatik und Mathematik
Email: *firstname.lastname*@uni-passau.de

***Abstract*—Modeling devices has become a crucial task in the Internet of Things (IoT) and Semantic Web technologies are seen as a promising tool for this purpose. However, as it may be arduous to manipulate semantic models, industrial solutions often re-define non-standard, simplified semantics. This is the case with Project Haystack, a framework to tag devices with labels from a predefined vocabulary in the field of Building Automation.**

In order to make Project Haystack standard and fully semantic, we wrapped its vocabulary in an ontology. In this paper, we present the general structure of this ontology, along with a method to turn tag sets into a Semantic Web model and back. The whole results in a reusable ontology design pattern. We aligned our Haystack tagging ontology with the wide-spread Semantic Sensor Network upper ontology and we designed a configuration environment for Building Automation systems based on semantic data, so as to discuss the added-value of semantics in automation.

I. Introduction

The trend of the Internet of Things (IoT) focuses on designing systems with small and affordable devices capable of pervasive computation, such as sensing devices or remotely controllable building equipments. It is based on the assumption that the number of IoT devices in the system is high.

While the aim of the IoT as a research field is to leverage the interplay between all those devices to create elaborate applications, it faces a major issue: IoT eco-systems are highly heterogeneous and require a lot of integration effort.

This is especially true in the context of Building Automation. Several incompatible yet equivalent standards emerged for Building Automation Systems (BAS), namely BACnet, KNX and LonWorks. Forced by the trend of the IoT, devices that until now controlled separate functions (e.g. lighting, ventilation or alarm functions) are now designed to communicate with each other, whatever communication protocol they use.

To address the problem of the interplay of heterogenous BAS, the need for a standardized generic model emerged, to describe what different Building Automation components may have in common. More precisely, there is currently no way to describe the *semantics* of those components regardless of their operational specifications. Project Haystack[1] is a recent initiative that aims at providing such a semantic representation.

[1]http://project-haystack.org/

Project Haystack can be seen from three different perspectives. It consists of:

1) a vocabulary of the domain of BAS, where each term is identified by a unique label (a Haystack tag),
2) a domain model giving a context for each tag and
3) a (loosely) REST API providing access to Haystack tags.

The founders of Project Haystack highlight particularly the fact that "[they] standardize semantic data models and Web services" (while they somehow put aside the tag perspective of the project). However, although it is popular and more and more adopted by the industry (thanks to its ease of use), Project Haystack lacks several features in the context of the IoT. Indeed, their data model exists for now as a text documentation and is not available in a formal representation. As the number of connected devices is supposed to continuously increase, it may raise scalability issues. Moreover, the proposed API to access Haystack tags does not follow widely-used Web standards (such as a REST interface or XML/JSON exchange formats). In particular, the only implementation available is hardly embedded-compliant. It requires computational power that most IoT devices of the future won't have (class I or II constrained devices, according to IETF terminology [5]).

In this paper, we re-define Haystack tags with the help of Semantic Web technologies (RDF, OWL, SPARQL) while keeping their ease of use and flexibility. We aim at keeping the benefits of a tagging logic by holding all tags, augmented with a domain ontology to formally describe the standard relations between the tags, plus a transformation method to turn tags into graph knowledge and vice versa.

Our re-definition allows for more automated processing of BAS data and exchange schemes based on well-known technologies. We discuss in Section II the benefits to formally represent semantics in the domain of Building Automation and, more generally in the IoT, by reviewing previous works on this topic. The ontology we designed is then presented in Section III. Our contribution is the ontology design pattern we created for this ontology. A prototype is presented in Section IV so as to expose in practice the benefits of semantics we previously identified.

II. Related Work

Numerous works have already been conducted in the field of semantic models for automation. Most of them have been driven by IoT requirements. More generally, semantics has been identified as one of the three main perspectives of the IoT paradigm (along with network and hardware) [4]. Although we refer here to "semantic" technologies, it would be more accurate to speak about "Semantic Web" technologies since all applications we present here are exclusively implemented using the semantic stack standardized by the W3C[2].

We distinguished between two kinds of requirements that led to using Semantic Web technologies: *interoperability* and *automatic processing*. We argue that a formal semantic representation of Project Haystack brings a contribution to both topics.

A. Interoperability

It is stated in its documentation that Project Haystack "facilitates 'mapping' of Haystack semantic tagging with other relevant standards" [1]. However, Semantic Web technologies and the principle of Linked Data have been already used for a couple of years for this purpose.

About the Semantic Web as an interoperability technology, one remarkable contribution to the IoT was the definition of the W3C Semantic Sensor Network (SSN) ontology [7]. This ontology gives a model to describe sensor meansurement capacities and observation data (among others). The guidelines of such an ontology date from 2008: a semantic sensor ontology should help attach spatial, temporal and thematic metadata to raw sensor observations [20]. The WGS84 vocabulary[3] and the OWL-Time ontology[4] were identified as possible formats for spatial and temporal information (respectively). With the help of the Semantic Web stack, sensor data can be then published in the Linked Open Data cloud and used by any high-level application, regardless of the data acquisition layer [14].

Since its definition, the SSN ontology has been widely used in IoT projects. SPITFIRE, that can be regarded as the flagship project for a semantic IoT, developed an ontology that heavily relies on SSN [15].

Before that time, other uses of Semantic Web technologies were reported in the context of Building Automation. For instance, an OWL ontology was proposed to unify the application models of BACnet, KNX, LonWorks and ZigBee [17]. Such a generic BAS model has the benefit that the mapping between the ontology and those protocols has only to be done once for each protocol. Once the mapping exists, a system engineer could configure all platforms with a single tool while protocol-specific parameterization is automatically infered by formal reasoning.

Similarly, the Ontology Device Description framework [9] was about separating upper knowledge from manufacturer-specific information about BAS devices. However, this work was a competitor to SSN and is now of lower interest since the latter is about to be standardized.

Another example is an expert system for BAS requirement elicitation that includes a domain model based on OWL [18]. This model is composed with rules expressed in SWRL, a rule markup language that integrates with OWL.

B. Automatic Processing

Formal semantics has always come along with techniques for automatic processing based on declarative knowledge, i.e. *reasoning* over this knowledge. Expert systems, as in the previous example, are one example of such processing. More recently, as the IoT demands "smarter" devices, new applications for logic reasoning emerged. The Semantic Web is then seen as an enabler for autonomous computing.

OWL, the Web Ontology Language[5], revolves around the theory of Description Logic (DL). Provided this formal framework, known inference methods can be used to generate (or *entail*) new pieces of knowledge (or *axioms*) from existing models. This principle was used to automatically compose services advertized by KNX devices annotated with DL knowledge [19].

DL reasoning can also be used to check the consistency of a given model. It has been successfully used for plant model validation [2]. The dedicated markup format for plant engineering (called CAEX) had to be mapped to an OWL ontology first.

To a lesser extent, the formalism of OWL was also used in a recently published Fault Detection and Diagnosis system for BAS [16]. In this case, knowledge inference is not done by reasoning whereas by updating existing knowledge that matches predefined graph patterns. graph patterns are expressed as SPARQL queries (or more precisely its counterpart to update graph knowledge, SPARQL Update[6], also called SPARUL). The infered knowledge corresponds to diagnoses that are delivered when anomalies are detected. The sensors are described by means of the SSN ontology.

SPARQL, as a standard query interface for semantic data[7], is another tool to automatically process sensor data. As mentioned previously, SPITFIRE describes sensors and sensor data with an ontology. As a consequence, it is possible to query this data and act accordingly. Sensor data streams could even be processed and queried on-the-fly, as events occur [3].

The architecture of SPITFIRE (already mentioned in the previous section) involves sensors carrying their own semantic data in a distributed fashion and exposing them through a Web-based REST interface. This is at the basis of the vision of the Web of Things (WoT), which shows many benefits. Automatic service discovery is one of them [6], where SPARQL also plays a significant role.

To summarize, the Semantic Web has become an integral part of the IoT, notably because of the features we pointed out above. In comparison, Project Haystack, while focusing on similar issues, does not cover all those features with its own specification.

[2]http://www.w3.org/2001/sw/

[3]http://www.w3.org/2003/01/geo/wgs84_pos

[4]http://www.w3.org/TR/owl-time/

[5]http://www.w3.org/TR/2012/REC-owl2-overview-20121211/

[6]http://www.w3.org/TR/2013/REC-sparql11-update-20130321/

[7]http://www.w3.org/TR/2013/REC-sparql11-query-20130321

As an illustration, Fault Detection models for BAS can be classified in three categories, as mentioned in [16]. As a BAS model, Project Haystack falls in the first category, namely "physical-based models", regarded as the one including the most accurate and exhaustive anomaly detection mechanisms. However, because Project Haystack lacks a formal model representation and a more standard API, advanced automatic building diagnosis based upon it is hardly conceivable.

While we have conciseness and ease of use from one side and interoperability and automatic processing enablement from the other side, we argue that we can combine both aspects with little effort. To this end, we "semantified" Haystack model, i.e. we created an ontology over the existing Haystack model to extend its expressiveness. We called this ontology the Haystack Tagging Ontology.

III. Haystack Tagging Ontology

Project Haystack was designed to help system engineers or domain engineers tag entities involved in a BAS environment. Entities can be automation devices (such as sensors, actuators or controllers) or building equipment, including Heating, Ventilation and Air Conditioning (HVAC) assets, lightings or energy meters.

A tag-based model is intended to be flexible. The tagger is free to combine any tags of their choice. However, most of Haystack tags have been designed so that they should be combined only in certain ways. Thus, there exists standard tag combinations corresponding to well-known building equipment. That is, there is also a pre-defined domain model behind Haystack tags, as mentioned in the introduction. The Haystack Tagging Ontology (HTO), presented in this paper, is an attempt to formalize this model. An overview of the ontology is given in Figure 1.

A. Design Pattern

Although a formal representation has several benefits, the tag structure of Project Haystack has been motivated by a clear requirement: it should provide a BA domain model that is quickly understandable and easy to use. Indeed, the growing complexity of BAS architectures, generating more and more data, should be hidden from the domain engineer that has in general limited knowledge about information technologies.

Moreover, Haystack tags have the benefit of being concise. They could be locally stored and shared directly between field devices. In the context of the WoT that [15] and [6] embody, it may be of interest to keep Haystack tag structure to let embedded devices describe themselves.

Therefore, our ontology tries to combine the benefits of a tag representation (ease of use, conciseness of tags) with those of a formal representation (interoperability, automatic processing). To this end, HTO presents a novel ontology design pattern (in the sense of [10]) that makes both representations possible, along with a transformation method between them.

This design pattern consists in separating the vocabulary part (raw tags) from the ontological part (types and relations). Vocabulary and ontological relations are made consistent with each other thanks to a common meta-model. Figure 1 shows the 3 resulting fragments (although they cannot be represented in OWL): vocabulary, domain model and meta-model.

In this design pattern, the domain model and the vocabulary are aware of the meta-model, i.e. their entities subsume or reference those of the meta-model. The contrary should not hold. What is more, the vocabulary annotates the domain model while the latter is not aware of the vocabulary. The presence of a meta-model makes possible a model transformation between the vocabulary and the domain model since they can refer to each other (see Section III-C).

In the following, we first introduce how Haystack tags were integrated into HTO, that is, the vocabulary part. Then, we present the domain model and the transformation method that our design pattern enables.

B. Haystack Vocabulary RDF Representation

As one can see in Figure 1, all Haystack tags are of type `HTag`. In OWL terminology, the latter is called a class while tags are individuals. As mentioned previsouly, Haystack tags are used to tag BAS entities, hence the definition of the class `HEntity` and the relation `hasTag` linking `HEntity` with `HTag`. A relation in OWL is called a property.

Classes, individuals and properties are all uniquely identified as Semantic Web resources, that is, they are given a URI composed of a namespace (we arbitrarily chose `<http://project-haystack.org/hto#>`) and a local name (represented in the diagram). For instance, the tag `sensor` is actually fully identified by `<http://project-haystack.org/hto#sensor>` (or in its short form, `hto:sensor`). It is especially useful for tag individuals, so that they can be unambiguously referred to.

To link entities with each other (for instance, sensors with the building equipment it belongs to), Haystack uses references, wich are a special kind of tag including the name of the referenced entity with tag label (as a key/value pair). So as to be referenced, Haystack entities must have an `id` tag, also expressed as a key/value pair. In HTO, we modeled references with the property `hasRef` while `id` is not required anymore since we use URIs.

For human readability, Haystack entities may also declare a textual description (using the tag `dis`). The Semantic Web already standardized powerful textual annotation tools (e.g. using Dublin Core vocabulary). In HTO, we simply replaced the tag `dis` with the property `rdfs:label`[8], which is the simplest way to annotate a Semantic Web resource.

All this put together, the proposed Semantic Web representation for tags is a simple RDF document declaring an `HEntity` with one or more relations to `Htag`s and/or to other Haystack Entities. A relation involving two elements linked together by a property is called a triple.

Let us assume we have a data point measuring temperature where air from a Air Heating Unit (AHU) is supplied (or discharged) to a room. This data point would be modeled with the following tags in Haystack:

[8] http://www.w3.org/TR/rdf-schema/\#ch_label

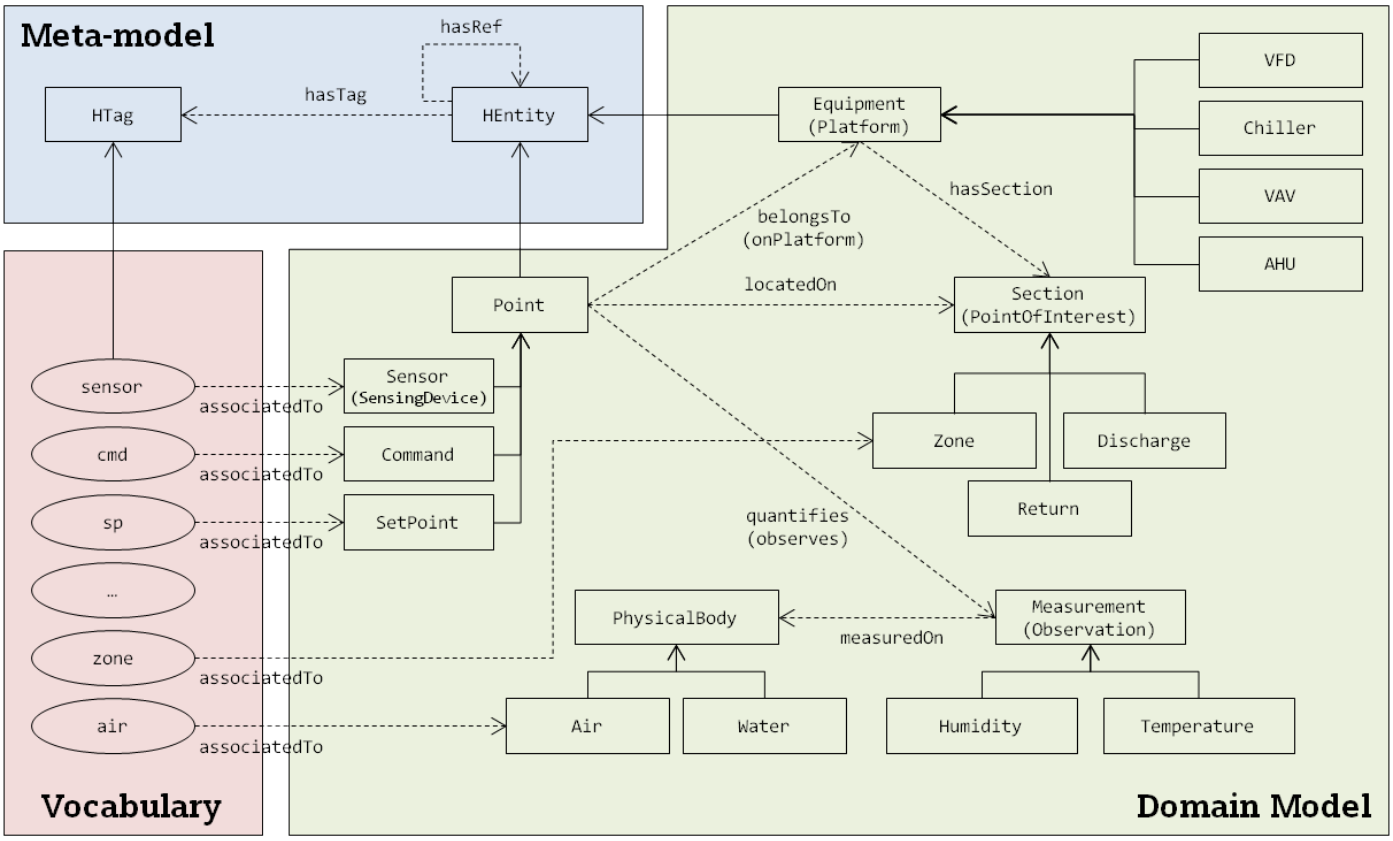

Fig. 1. Overview of the Haystack Tagging Ontology. OWL entities in the vocabulary are all individuals, while other entities are classes (namespaces were omitted). Names in parenthesis indicate equivalent entities from SSN. For sake of simplicity, only a few subclasses are represented.

Listing 1. A Haystack-tagged entity (Zinc syntax)

```
id: @point01254
dis: "point01254"
sensor
temp
discharge
equipRef: @equip07454
```

would then have the following RDF triples (we assumed that all entities are defined in the universal example namespace `http://example.org#`, short `ex:`):

Listing 2. RDF representation of entity of Listing 1 (N3 syntax)

```
ex:point01254 rdf:type hto:HEntity;
              rdfs:label "point01254";
              hto:hasTag hto:sensor;
              hto:hasTag hto:temp;
              hto:hasTag hto:discharge;
              hto:hasRef ex:equip07454.
```

It is worth noting here that we did not consider the whole set of Haystack tags. There exists other kinds of key/value tags for numbers, strings, etc. Similarly to `rdfs:label`, RDF provides better ways to express such values. It includes for instance a full standardized type hierarchy (XML Schema datatypes). Though, the subset we considered represents 86% of Haystack model (145/169 tags).

Moreover, Project Haystack also intends to model the whole building so that its location can also be tagged. Again, the Semantic Web already has extensive tools to handle geographic data, so we did not include sites in HTO. The WGS84 vocabulary cited by [20] and GeoSPARQL[9], the SPARQL extension for geograhical data standardized by the Open Geospatial Consortium (OGC), are better candidates for this purpose.

A substantial benefit of having an RDF representation of Haystack tags is that they can be easily exchanged and consumed through a standard interface. Project Haystack also identified the need for easy tag consumption; it specifies its own interface to read tags and navigate from one entity to another.

Regarding the navigation, the Semantic Web makes it significantly simple thanks to URIs: from `point01254`, one simply has to follow the URL of `equip07454` to navigate further (i.e. `<http://example.org#equip07454>`). This is the core idea of Linked Data[10], a well-known part of the Semantic Web already explored in [14] for sensor data.

In practice, Haystack tags are often stored in a central repository running a single Haystack server. In this case, reading tags in RDF can be made through SPARQL. This standard Semantic Web query language specifies how to search for graph pattern, filter values or combine knowledge in a very expressive way. Additionally, two HTTP interfaces

[9] http://www.opengeospatial.org/standards/geosparql
[10] http://www.w3.org/DesignIssues/LinkedData.html

have been standardized to submit SPARQL queries. All the features intended to be standardized by Project Haystack's API (type definition, query interface, filters and navigation) can be superseded by SPARQL.

In the case where the tags are carried by field devices, the limited device computational power prevents SPARQL to be used directly. It is still possible to retrieve them as a Linked Data document, though. Effort has been recently made to bring the semantics down to the field device (as part of the self-reporting paradigm of the WoT) [11], [13]. These approaches propose efficient and compact binary formats for RDF. A Semantic Web search interface for embedded devices is part of future work.

C. Domain Model and Transformation

Representing Haystack tags in RDF presents the benefit of using standard and already implemented Web technologies. However, as such, the contribution in automatic processing remains poor. To address this second issue, HTO has to provide a Semantic Web representation of the model behind Haystack tags.

Haystack provides a *domain* model. It describes general BA concepts regardless of how they are used in a given BAS configuration. A domain model is intended to be re-used and extended by an *application* model according to specific needs. In most ontology engineering methods, domain knowledge is often captured in the following successive forms: first as a lexicon of domain-specific notions, second as a glossary (previously identified notions are enriched with definitions) and third as a semantic network, or knowledge graph, linking glossary elements with each other [8].

Project Haystack already built a freely available and well-documented glossary; one could bring it to the semantic level with little effort. Our HTO domain model ontology is rather small. It contains 58 classes, 10 properties and 62 individuals, for a total of having 128 OWL axioms (as a comparison, SSN has 644 axioms). Looking at Figure 1, one may note that all the individuals defined in HTO are tags, which means that we included roughly half of the whole tag set (references do not count).

Roughly, our proposed model defines ontological properties between high-level classes (`Point`, `Section`, `Equipment`, `Measurement` and `PhysicalBody`) and a taxonomy for each of these classes. The central classes are `Point` and `Equipment`. The former models data points, i.e. automation devices that produce or consume data while the latter models any kind of building equipment that is automated. Equipments are further splitted in `Sections`, modeling e.g. ductwork, condenser or heat wheel.

Having high-level classes greatly facilitates querying. For instance, it would be easy to formulate queries such as "find all equipments" or "find everything that is quantified for a given equipment" (the latter is part of the SPARUL query in Listing 11). Such queries are hardly expressible with raw Haystack tags. For instance, finding all equipments would require the use of each equipment tag (`vav`, `ahu`, `chiller`, etc).

So as to combine the ease of use of Haystack tags and machine processing capabilities of the Semantic Web, we designed the ontological model so that an automatic transformation is possible from the vocabulary to the domain model.

For each tag from the vocabulary, we defined an OWL class in the domain model. The mapping between them is indicated by an annotation property: `hto:associatedTo`, which is the cornerstone of the design pattern used in HTO. As an example, the ontology contains the axiom `hto:temp hto:associatedTo hto:Temperature`. Therefore, if a given entity is tagged with `temp`, the domain model should reflect it by having an individual of type `Temperature`. We can fomulate this transformation rule the following way:

Listing 3. First HTO transformation rule (N3 syntax)

```
@forSome :i.
{ ?e hto:hasTag ?t.
  ?t hto:associatedTo ?c. }
=>
{ :i a ?c. }.
```

The implication notation of N3 syntax should be read as follows: for each triple pattern in the antecedent (upper part) where variables `?e`, `?t` and `?c` can be bound with existing individuals (respectively an entity, a tag and a class), the pattern in the consequent (lower part) should hold. In practice, in order to match this rule, `?i` is bound to a new individual, whose local name is arbitrarily chosen (in our examples, the tag label and a random four-digit number are concatenated).

In the particular case where the OWL class bound to `?c` is a subclass of `HEntity`, `?i` should be bound to the individual that carries tags — which already exists. This could be done by asserting that the created individual and the existing one are strictly equivalent, using `owl:sameAs`.

If we apply this rule on our previous example of Section III-B, we obtain the following individuals:

Listing 4. Result of the 1st rule applied on example of Listing 2

```
ex:sensor03588 a hto:Sensor.
ex:temp05488 a hto:Temperature.
ex:discharge06954 a hto:Discharge.
ex:point01254 owl:sameAs ex:sensor03588.
```

The second transformation rule applies to Haystack references, which are simply translated into the equivalent notion in the domain model. It is as follows:

Listing 5. Second HTO transformation rule (N3 syntax)

```
{ ?e hto:hasRef ?ref }
=>
{ ?e hto:belongsTo ?ref. }.
```

The reference declared by `point01254` becomes:

Listing 6. Result of 2nd rule applied on example of Listing 2

```
ex:point01254 hto:belongsTo ex:equip07454
```

A third transformation rules is used to add links between the individuals created by the first rule (Listing 3). It is based on the assumption that all properties of the domain model define a source class and a target class. I.e. the domain and the range of the properties have to be asserted. We designed HTO

accordingly, which gives us the possibility to infer the right relation between individuals by comparing the domain and the range of available properties in the ontology, as follows:

Listing 7. Third HTO transformation rule (N3 syntax)

```
{ ?i1 a ?c1.
  ?i2 a ?c2.
  ?p rdfs:domain ?c1;
     rdfs:range ?c2. }
=>
{ :i1 ?p :i2 }
```

The properties infered by this rule in our example are the following:

Listing 8. Result of 3rd rule applied on example of Listing 2

```
ex:point01254 hto:quantifies ex:temp05488;
              hto:locatedOn
                ex:discharge06954.
ex:equip07454 hto:hasSection
                ex:discharge06954.
```

If we successively apply the three rules we have just presented, we are able to generate the graph knowledge summarized in Figure 2. The reverse transformation is trivial. To retrieve the Haystack tags that are associated to a given `HEntity`, we gather all individuals that are in its neighborhood in the knowledge graph, recursively, until either another `HEntity` is reached or the neighborhood is empty. The domain model should then be acyclic, which holds if it is a simple directed graph (per definition).

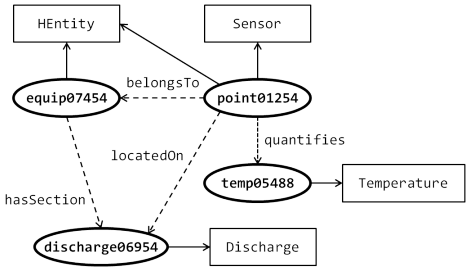

Fig. 2. Graph knowledge generated from the tagged entity of Listing 2 (namespaces were omitted).

IV. Prototyping & Discussion

As mentioned earlier, the concepts we present here, such as our ontology design pattern and the automatic transformation rules have been prototyped in the form of a small OWL ontology. We performed several tests on this ontology using the standard tag combinations provided by Project Haystack[11]. We generated a test case with one building containing the four main HVAC equipments modeled by Haystack (`ahu`, `vav`, `vfd` and `chiller`), equipped with all data points that correspond to a standard combination.

We implemented the model transformation using the OWL API [12] combined with Pellet reasoner [21]. Recently, SPARUL has been standardized as another tool to manipulate RDF data. It is actually an interface that takes SPARQL syntax to create, update or delete data in semantic repositories. However, SPARUL is not tied to OWL. It does not easily integrate DL reasoning tools. Because we intend to benefit from DL reasoning for automatic processing, we discarded SPARUL for this implementation.

[11] http://project-haystack.org/download\#equip-points

A. Reversibility

We first tried to assert that HTO captures the semantics of standard combinations defined by Project Haystack. A way to do it is to the check that the transformation we proposed is reversible for all of those combinations. Starting from a tag representation of the example we built with standard tag combinations, we turned it into a knowledge graph and back (with the reverse transformation we also briefly mentioned) and compared the generated tags with the original ones.

Among the 85 standard data points defined by Project Haystack for the main HVAC equipments, our prototype covers 67 of them (the remaining includes tags that have not been modeled yet). As expected, all combinations could be reconstructed after the transformation.

However, for arbitrary tag combinations, the reversibility can not be guaranteed. Indeed, not all possible co-occurences are modeled by the domain model of HTO. Our prototype models about 200 co-occurences while there are 3782 possible tag pairs in total (62×61). If a new tag combination is given, there may exist no suitable property to link the created individuals (Listing 7). As the size of the model will grow (Project Haystack is still an on-going activity), the problem may become critical.

The definition of a strict domain and range for each property makes the domain model rather fixed and the domain knowledge that is automatically asserted may became erroneous if tagging usage is changing over time. However, if BAS engineering and provisioning are assisted with ontologies and automatic tools built upon it, this situation is not likely to happen.

B. Consistency Check

While the fact that tagging usage may change over time is unlikely, it is still possible to use tags in a inconsistent manner. Formal logic helps prevent this from happening thanks to automatic consistency checking. For instance, one may tag a single data point with both `sensor` and `cmd`, misinterpreting the role of data points in the system. In this case, a DL reasoner will be able to detect this inconsistency since the classes `Sensor` and `Command` are disjoint in the ontology: an entity is not allowed to belong to those two classes in the same time.

In our implementation, the consistency of the graph knowledge generated from tags is always checked. We ensured that the overall system in our test case, containing 50 sensors and 22 actuators (or commands) and setpoints, makes sense. We even use consistency checking to solve particular cases where Haystack tags are inherently ambiguous.

For instance, the tag `fan` can be modeled either as a section of an equipment (`Fan`) or as a whole equipment `VFDFan`), depending on co-occurring tags. In HTO, `fan` is associated to

both classes. At the time of the transformation, the generated knowledge is detected as inconsistent since the tagged entity is asserted both types, which are also declared as disjoint (all high-level classes are disjoint).

As a result, changes are dismissed and our implementation checks types one after the other. If `fan` co-occurs with `cmd`, then the tagged entity typed `VFDFan` leads to an inconsistency and the tag is finally associated to `Fan`, a section of an Air Heating Unit or a Variable Air Volume system. This example illustrates how formal modeling can meet tagging flexibility.

C. BAS Provisioning

HTO focuses on bringing interoperable models and enabling automatic processing in the different phases of a BAS lifecycle. About interoperability, the Semantic Web facilitates alignment between ontologies and semantic data sets. For instance, it is possible to integrate the SSN ontology (mentioned in Section II-A) into HTO so as to unify sensor data.

In the test case we built, there is only one HVAC system managing a single room (modeled as a `zone` in Haystack). If one wants to share the room temperature to external high-level applications, one could easily take advantage of the existing model to generate SSN data. As an example, Pellet DL reasoner can infer SSN data in Listing 10 from HTO data in Listing 9.

Listing 9. HTO representation of a room temperature sensor

```
ex:point01254 a hto:Sensor;
              hto:quantifies ex:temp05648;
              hto:locatedOn ex:zone09655.
ex:temp05648 a hto:Temperature.
ex:zone09655 a hto:Zone.
```

Listing 10. SSN/QU representation of a room temperature sensor

```
ex:point01254 a ssn:SensingDevice;
              ssn:observes temp05648;
ex:temp05648 a qu:Temperature;
             ssn:isPropertyOf ex:zone09655.
ex:zone09655 a ssn:FeatureOfInterest.
```

The equivalence can be purely expressed in OWL (using `owl:equivalentClass` or `owl:equivalentProperty`) and generated by a DL reasoner. We defined several alignments in HTO as indicated in Figure 1. In this particular example, we used the W3C ontology for Quantity Kinds and Units (QU) to get a SSN-compliant definition of temperature[12]. Once a data point is expressed using SSN/QU, one can simply add temperature values using `ssn:SensorOutput`.

D. BAS Engineering

As a last example, we focus on automatic processing. With the help of a SPARUL query, the different equipments of our test building are turned into "functions" with inputs and outputs. This offers a view where components can be composed with each other so that sensors and commands/set-points directly interact with each other. Since their definition is formal, one could easily think of an automatic composition of these functions by means of DL reasoning or targeted SPARQL querying, as in [19] or [16].

Though, BAS automatic processing was not the scope of the present paper. Instead, we included HTO in an engineering tool to manually compose BAS functions. The tool we use here is a flow-based configuration environment. I.e. functions are represented as nodes and data flow as edges that connect functions with each other. The resulting configuration can then be seen as a graph.

To integrate HTO into this tool, we applied the query given in Listing 11 on the semantic data we generated for our test building. As an interface, we created a simple ontology (prefixed by `fbc:`, which stands for Flow-based Configuration) that define `Component` (i.e. node or function) and `Input` and `Output`.

Listing 11. SPARUL query turning Haystack entities into flow-based components

```
INSERT {
  ?eq a fbc:Component;
      fbc:hasOutput ?measout;
      fbc:hasInput ?measin.
} WHERE {
  ?out a hto:Sensor;
      hto:quantifies ?measout;
      hto:belongsTo ?eq.
  {
    ?in a hto:Command;
      hto:quantifies ?measin;
      hto:belongsTo ?eq.
  } UNION {
    ?in a hto:SetPoint;
      hto:quantifies ?measin;
      hto:belongsTo ?eq.
  }
}
```

Once the query is executed, the building data can be visualized (see Figure 3).

This idea of flow-based configuration or automatic composition was already presented in [17] and [9]. The latter presents an ontology for BAS controllers, which defines controller profiles (equivalent to components). Data points attached to a controller profile also form inputs and outputs. However, contrary to us, they separate the notions of component and function: a controller profile can implement several functions.

This way, they follow IEC specification of function blocks (IEC 61499) to model distributed industrial systems. This point of view was motivated by the presence of controller devices between field devices and other infrastructures. However, as a consequence of the IoT trend, controllers tend to disappear now that field devices become powerful enough to handle data themselves. Our example tries to adopt a more IoT-oriented flow-based model.

V. Conclusion

The present paper presented an ontology for Project Haystack domain model in order to be able to combine

[12] http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu-rec20.html

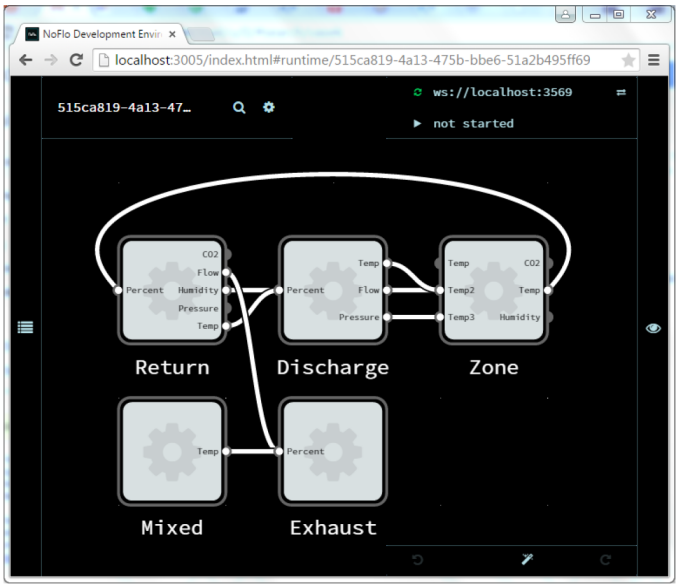

Fig. 3. Flow-based configuration environment that let communication between field devices be visually parameterized (on this example, connections have no precise meaning). The components represented here were automatically generated from the sections of the Air Heating Unit of our test case. Inputs and outputs are data points.

proven Semantic Web technologies (SPARQL querying, DL reasoning) with Haystack tags. The ontology design pattern we applied, while providing a formal representation, also keeps the benefits of a tag representation (i.e. conciseness and ease of use): the prototype we modeled allows a reversible automatic transformation between tags and graph knowledge for most of the standard tag combinations.

However, several limitations have been identified in the discussion. First, although they have been identified as a key element in the context of the Web of Things, Semantic Web technologies hardly scale to embedded environments. We previously mentioned that field devices cannot run a whole SPARQL endpoint to search their semantic data, in which case it should be substituted by a static Linked Data document. Moreoever, as we noted during our experiments about consistency checking, DL reasoners may also cause scalability issues since they require high computational power. As we embrace the vision of the Web of Things, designing a light version of the Semantic Web is part of our future work.

Second, because Project Haystack is an on-going activity and is destined to evolve, an ontology may not perfectly fit. In particular, tagging usage may change over time, which would not be reflected by the ontological model. Still, if the transformation method we proposed does not meet the need of the industry, the RDF exchange format for Hastack tags we presented as a first contribution remains of interest as an interoperabilty and extensive annotation enabler. A key element is its alignment with SSN/QU.

References

[1] Guide specification for data modeling of building systems and equipment based on project haystack open source data modeling standard. http://project-haystack.org/download/file/Guide-Specification.docx, February 2014.

[2] L. Abele, C. Legat, S. Grimm, and A. Muller. Ontology-based validation of plant models. In *11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 236–241, July 2013.

[3] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. Ep-sparql: A unified language for event processing and stream reasoning. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 635–644, 2011.

[4] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[5] C. Bormann, M. Ersue, and A. Keranen. Terminology for constrained-node networks. RFC 7228, May 2014.

[6] G. Bovet and J. Hennebert. Distributed semantic discovery for web-of-things enabled smart buildings. In *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5. IEEE, March 2014.

[7] M. Compton, P. Barnaghi, L. Bermudez, R. Garca-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25–32, 2012.

[8] A. De Nicola, M. Missikoff, and R. Navigli. A software engineering approach to ontology building. *Information Systems*, 34(2):258–275, April 2009.

[9] H. Dibowski and K. Kabitzsch. Ontology-based device descriptions and device repository for building automation devices. *EURASIP Journal on Embedded Systems*, 2011(1):623461, April 2010.

[10] A. Gangemi. Ontology design patterns for semantic web content. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *The Semantic Web ISWC 2005*, pages 262–276. Springer Berlin Heidelberg, 2005.

[11] H. Hasemann, A. Kroller, and M. Pagel. RDF provisioning for the internet of things. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 143–150, 2012.

[12] M. Horridge and S. Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic web*, 2(1):11–21, January 2011.

[13] S. Kaebisch, D. Peintner, and D. Anicic. Standardized and efficient RDF encoding for constrained embedded networks. In *2015 12th European Semantic Web Conference (ESWC)*, June 2015.

[14] H. Patni, C. Henson, and A. Sheth. Linked sensor data. In *2010 International Symposium on Collaborative Technologies and Systems (CTS)*, pages 362–370, 2010.

[15] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Krller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant, and R. Richardson. SPITFIRE: toward a semantic web of things. *IEEE Communications Magazine*, 49(11):40–48, November 2011.

[16] J. Ploennigs, A. Schumann, and F. Lcu. Adapting semantic sensor networks for smart building diagnosis. In *The Semantic Web ISWC 2014*, pages 308–323. Springer International Publishing.

[17] C. Reinisch, W. Granzer, F. Praus, and W. Kastner. Integration of heterogeneous building automation systems using ontologies. In *34th Annual Conference of IEEE Industrial Electronics, 2008. IECON 2008*, pages 2736–2741, November 2008.

[18] S. Runde, A. Fay, and W.-O. Wutzke. Knowledge-based requirement-engineering of building automation systems by means of semantic web technologies. In *2009 7th IEEE International Conference on Industrial Informatics (INDIN)*, pages 267–272, June 2009.

[19] M. Ruta, F. Scioscia, E. Di Sciascio, and G. Loseto. Semantic-based enhancement of ISO/IEC 14543-3 EIB/KNX standard for building automation. *IEEE Transactions on Industrial Informatics*, 7(4):731–739, November 2011.

[20] A. Sheth, C. Henson, and S. S. Sahoo. Semantic sensor web. *IEEE Internet Computing*, 12(4):78–83, 2008.

[21] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.